

Logic Programming Engineering – Assignment 5

Dr.-Ing. Sascha Klüppelholz

Exercise 5.1 [Simple arithmetic on lists]

- a) Define a predicate `maxmember/2` that returns the maximum of the elements of a list.

Solutions:

```
maxmember([H],H) :- !.
```

```
maxmember([H|Tail],Max) :-  
    maxmember(Tail,M),  
    (H>M -> Max is H; Max is M).
```

- b) Write a predicate `average/2` which computes the average over the numeric elements of a list. For this write an auxiliary predicate `sum/2` for the sum of elements of a list.

Solutions:

```
sum([X],X) :- !.
```

```
sum([H|Tail], Sum) :-  
    sum(Tail, Sum1),  
    Sum is Sum1 + H.
```

```
average(List,Av) :- !,  
    length(List, N),  
    sum(List, Sum),  
    Av is Sum/N.
```

%%% a more efficient solution avoiding second run for length

```
sum_number([X],X,1) :- !.
```

```
sum_number([H|Tail],Sum, Number):-  
    sum_number(Tail, Sum1, Number1),  
    Sum is Sum1 + H,  
    Number is Number1 + 1.
```

```
average2(List,Av) :- !,  
    sum_number(List, Sum, Number),  
    Av is Sum/Number.
```

Exercise 5.2 [Sieve of Eratosthenes]

Provide a Prolog predicate `eratosthenes/2` that first generates all integers between 2 and a given upper bound and then eliminates all non-primes from the list. Your implementation should follow the idea of Eratosthenes Sieve¹. How many primes are there between 1 and one million? What is the greatest prime lower than one million?

Solution:

%%% naive solution

```
eratosthenes(N, L) :-
    findall(X, between(2, N, X), FullList),
    dosieve(FullList, 1, N, L).

dosieve(FullList, I, N, Result) :-
    nth1(I, FullList, Element),
    2*Element < N, !,
    createMults(Element, 2, N, Mults),
    ord_subtract(FullList, Mults, LPrime),
    IPrime is I+1,
    dosieve(LPrime, IPrime, N, Result).

dosieve(FullList, _, _, FullList).

createMults(I, Factor, MaxValue, [Element|Tail]) :-
    ((Factor+1)*I) =< MaxValue, !,
    Element is Factor*I,
    FactorPrime is Factor + 1,
    createMults(I, FactorPrime, MaxValue, Tail).

createMults(I, Factor, _, [Element]) :-
    Element is Factor*I.
```

%%% alternative solution

```
:- dynamic(composite/1), dynamic(isprime/1).

eratosthenes(N, L) :-
    retractall(composite/1),
    retractall(isprime/1),
    dosieve(2, N),
    bagof(X, isprime(X), L).

dosieve(Prime, N) :-
    assert(isprime(Prime)),
    markMults(Prime, 2, N),
    PrimeP is Prime + 1,
```

¹see, e.g., https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

```

between(PrimeP,N,NextPrime),

not(composite(NextPrime)) ->
dosieve(NextPrime, N)
; true.

markMults(I, Factor, MaxValue) :-
    ((Factor+1)*I) =< MaxValue, !,
    Element is Factor*I,
    ( composite(Element) -> true; assert(composite(Element)) ),
    FactorPrime is Factor + 1,
    markMults(I, FactorPrime, MaxValue).

markMults(I, Factor, MaxValue) :-
    Factor*I > MaxValue, !.

markMults(I, Factor, _) :-
    Element is Factor*I,
    ( composite(Element) -> true; assert(composite(Element)) ).

```

Output:

```

?- eratosthenes(100000,L), length(L,N), last(L,LP).
L = [2, 3, 5, 7, 11, 13, 17, 19|...], N = 78498, LP = 999983.

```

Exercise 5.3 [Subset sum]

Write a predicate `subsetsum/3` that computes for a given list `L` and a given integer number $n \in \mathbb{N}$ a sublist of `L` whose sum over all elements equals to `n`. The predicate should enumerate all solutions.

Solution:

```

subsetsum([H|_], H, [H]).

subsetsum([H|Tail], Sum, [H|STail]) :-
    Sum1 is Sum-H,
    subsetsum(Tail, Sum1, STail).

subsetsum([_|Tail], Sum, STail):-
    subsetsum(Tail, Sum, STail).

%%%
% alternative solution:
%%%

subsetsum2(L,N,M) :-
    mylist_subset(L,M),
    listsum(M,N).

```

```

listsum([E],E).

listsum([H|T],S) :-
    listsum(T,S1),
    S is H+S1.

% own subset implementation that enumerates all solutions
mylist_subset([], []).

mylist_subset([_| Tail], P) :-
    mylist_subset(Tail, P).

mylist_subset([Head | Tail], [Head | P]) :-
    mylist_subset(Tail, P).

```